December 11, 2024

Remote Workflows at ALCF

CHRISTINE SIMPSON
Assistant Computational Scientist
Data Services & Workflows Group
ALCF

Outline

- Motivation & Approach for Remote Compute
- Globus Compute: remote execution of functions
- Configuration of Compute Endpoints
- Globus Flows
- Conclusions

Triggering ALCF Workloads Remotely

Example Cases

- ALCF has an increasing number of workloads that are triggered from outside the facility
- Use cases include workloads from experimental facilities like X-ray light sources (APS), tokamak & neutrino experiments, and from LLM & inference services
- Remote triggering could be of use to any user seeking to coordinate work across multiple compute facilities or machines

1. Send data to ALCF
2. Run computation on Polaris
3. Return results to APS

DOE's Integrated Research Infrastructure (IRI) Vision:

To empower researchers to meld DOE's world-class research tools, infrastructure, and user facilities seamlessly and securely in novel ways to radically accelerate discovery and innovation

Advanced Networking

Edge Sensors

Advanced Computing

Researchers

Local Campus Computing

Computing Testbeds

Cloud Computing

Data Management

High Performance Data Facility

Data Repositories PuRE Data Assets

Software

Software and Applications

AI Tools Digital Twins

New modes of integrated science

Rapid data analysis and steering of experiments

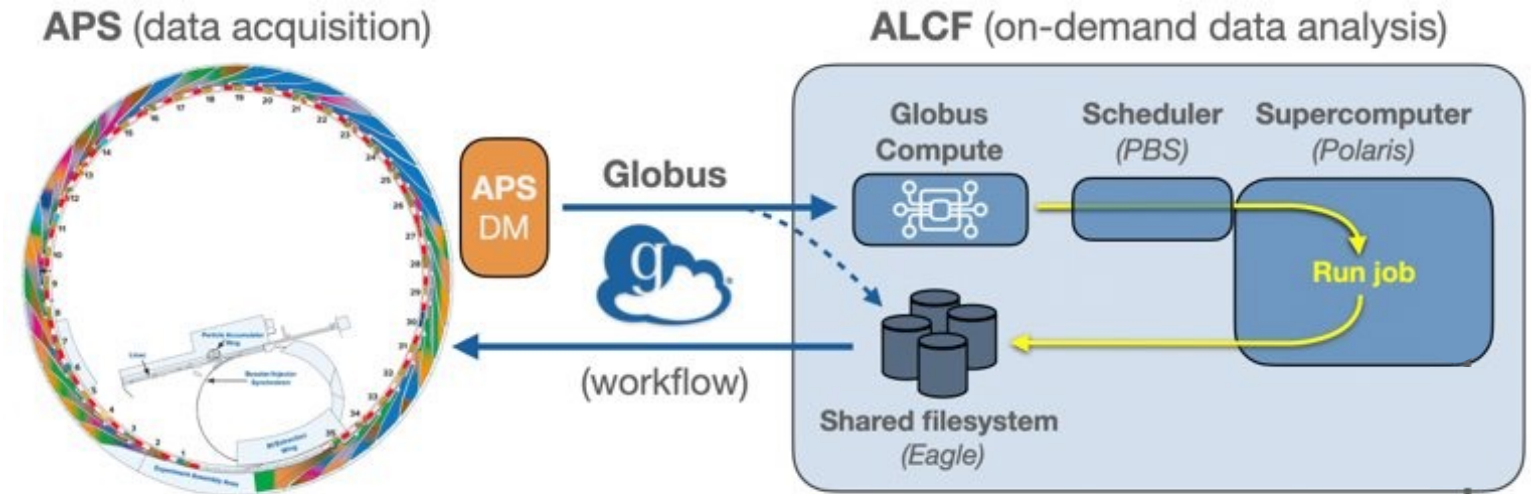
Novel workflows using multiple user facilities

AI-enabled insight from integrating vast data sources

U.S. DEPARTMENT OF ENERGY | Office of Science

Slide from Ben Brown, DOE, ASCAC, June 2023

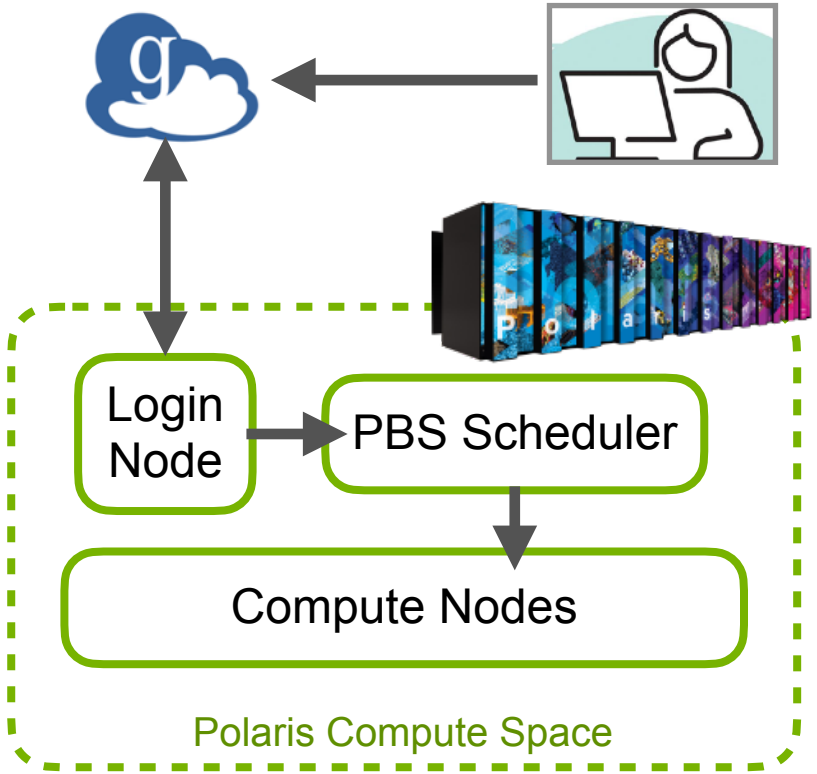
ARGONNE NEXUS: LIGHTSOURCE AUTOMATION



- Integration with the data management (DM) system at APS allows workflow to begin as soon as data is taken
- Workflow moves data from the APS beamline to ALCF and submits job to demand queue on Polaris
- Results are written to Eagle, where they're reachable via Jupyter, and also returned to APS for evaluation

Approach for Triggering Remote Work

- Challenge of triggering remote work is how to communicate with scheduler from outside the machine
- Approach: user starts a process on a login node that communicates with the scheduler and can reach out to a remote server/database to gather work tasks
- Globus Compute & Balsam are two services at ALCF that use this model for executing remote work

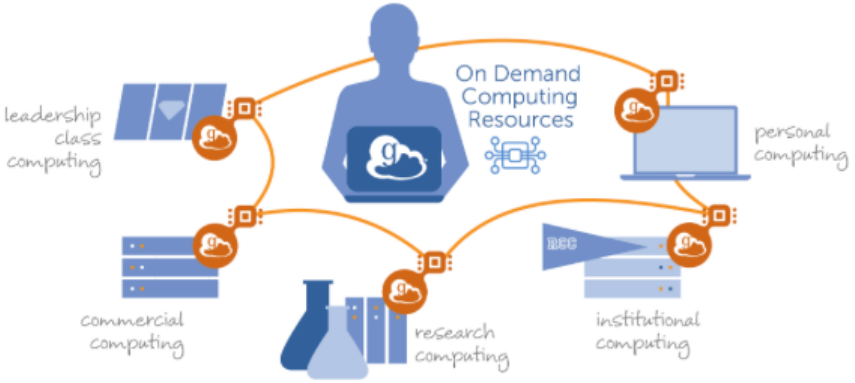


GLOBUS COMPUTE

Globus Compute

“fire and forget” execution of tasks

- Allows users to launch applications on ALCF systems remotely from a laptop, or any other external machine
- Executes **python functions**; compiled executables can be executed by wrapping them in a python function
- Requires setup of a **Compute Endpoint** on the target machine (e.g. Polaris) beforehand
- Built on top of Parsl, similar configuration
- Globus Compute functions can be integrated with data transfers with Globus Flows



Translating work to Globus Compute

PBS Batch Script -> Globus Compute Function & Endpoint

```
#!/bin/bash -l
#PBS -l select=1:system=polaris
#PBS -l place=scatter
#PBS -l walltime=0:30:00
#PBS -q debug
#PBS -A Catalyst
#PBS -l filesystems=home:grand:eagle

cd ${PBS_O_WORKDIR}

# Execute app on GPU 0
CUDA_VISIBLE_DEVICES=0 ./hello_affinity
```

- Example Case: Execute compiled application on a single GPU
- Goal: run this application many times, one task per GPU, remotely
- The PBS and resource options will translate to the **Globus Compute endpoint** on the Polaris Login Node
- The application call will translate to a **python function** sent from the remote machine

See Fall Workshop materials for hello_affinity example:
https://github.com/argonne-lcf/ALCF_Hands_on_HPC_Workshop/tree/master/workflows/globus_compute

Define Compute Function

Python function wraps executable

- Any compiled executables must be available on the target machine
- Any required libraries must be imported within the function
- Those libraries must be available in the environment running the endpoint

```
def hello_affinity_wrapper(run_directory):
    import subprocess
    import os

    # Create a run directory for the application to execute
    os.makedirs(os.path.expandvars(run_directory), exist_ok=True)
    os.chdir(os.path.expandvars(run_directory))

    # This is the command that calls the compiled executable
    command = "/path/to/hello_affinity"

    # This runs the application command
    res = subprocess.run(command.split(" "),
                         stdout=subprocess.PIPE,
                         stderr=subprocess.PIPE)

    # Write stdout and stderr to file on Polaris filesystem
    with open("hello.out", "w") as f:
        f.write(res.stdout.decode("utf-8"))
        f.write(res.stderr.decode("utf-8"))

    return res
```

From the Remote Machine: Send Functions to the Endpoint

- Endpoint ID for Compute Endpoint is needed
- Function can also be registered with Globus and run via a UUID (see workshop materials for example)

```python
from globus_compute_sdk import Executor

# First, define the function ...
def hello_affinity_wrapper(run_directory):
    ...see previous slide...
    return res

# Paste your endpoint id here
endpoint_id = '82e49eaa-3619-4b7c-963e-b020a16537fd'

# ... then create the executor, ...
with Executor(endpoint_id=endpoint_id) as gce:
    # ... then submit for execution, ...
    tasks = [gce.submit(hello_affinity_wrapper,
                        "$HOME/affinity/{i}")),
             for i in range(4)]

    # Wait for tasks to return
    for t in tasks:
        print(t.result())
```

```
> python submit_affinity_functions.py
```

On Polaris Login Node: Configure & Start Endpoint

Create environment and install:

```
> python -m venv my_env
> source source my_env/bin/activate
> pip install globus-compute-endpoint
```

Create endpoint:

```
> globus-compute-endpoint configure --endpoint-config /path/to/my_config.yaml my_endpoint
> globus-compute-endpoint start my_endpoint
> globus-compute-endpoint list
+--------------------------------------+--------------+----------------------+
|              Endpoint ID             |    Status    |     Endpoint Name    |
+======================================+==============+======================+
| 82e49eaa-3619-4b7c-963e-b020a16537fd | Running      | my_endpoint          |
+--------------------------------------+--------------+----------------------+
```

Endpoint Configuration

Polaris Example
1 GPU per worker

- Contents of `my_config.yaml ->`
- Contains all the information the endpoint process needs to submit jobs to PBS
- Defines resources assigned to each “worker”
- 1 “worker” runs 1 task at a time
- This example creates 4 workers per Polaris node, each pinned to a different GPU
- Different configs can be used for different worker-resource allocations, e.g. for multi-node MPI tasks

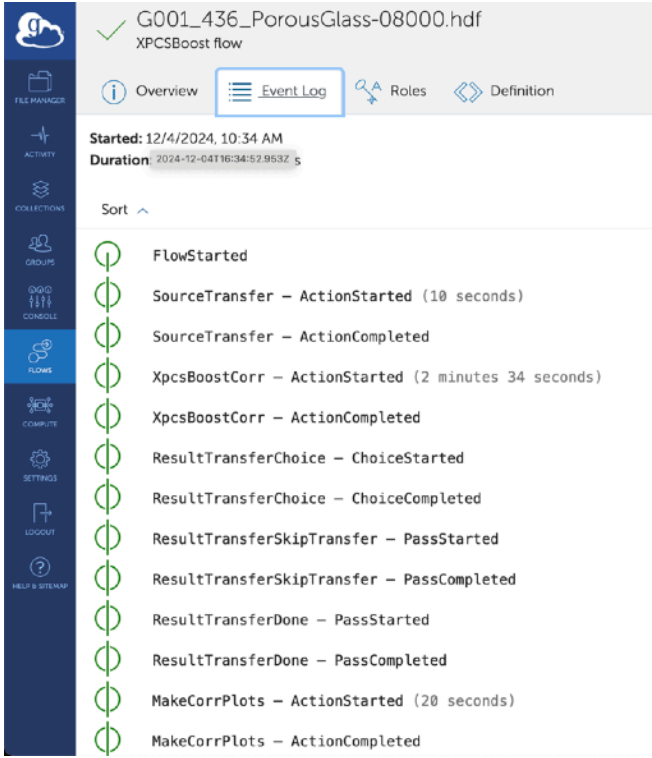
```yaml
engine:
    type: GlobusComputeEngine
    available_accelerators: 4 # Assign one worker per GPU
    max_workers_per_node: 4
    cpu_affinity: "list:24-31,56-63:16-23,48-55:8-15,40-47:0-7,32-39"
    prefetch_capacity: 0  # Increase for many more tasks than workers
    max_retries_on_system_failure: 2
    strategy: simple

    job_status_kwargs:
        max_idletime: 300
        strategy_period: 60

    provider:
        type: PBSProProvider
        launcher:
            type: MpiExecLauncher
            # Ensures 1 manger per node, work on all 64 cores
            bind_cmd: --cpu-bind
            overrides: --ppn 1
        account: Catalyst
        queue: debug
        cpus_per_node: 64
        select_options: ngpus=4
        scheduler_options: "#PBS -l filesystems=home:eagle:grand"

        # Node setup: activate environment running endpoint
        worker_init: “source /path/to/my_env“
        walltime: 00:30:00

        nodes_per_block: 1 # nodes per PBS job
        init_blocks: 0
        min_blocks: 0
        max_blocks: 1 # maximum number of PBS jobs
```

GLOBUS FLOWS

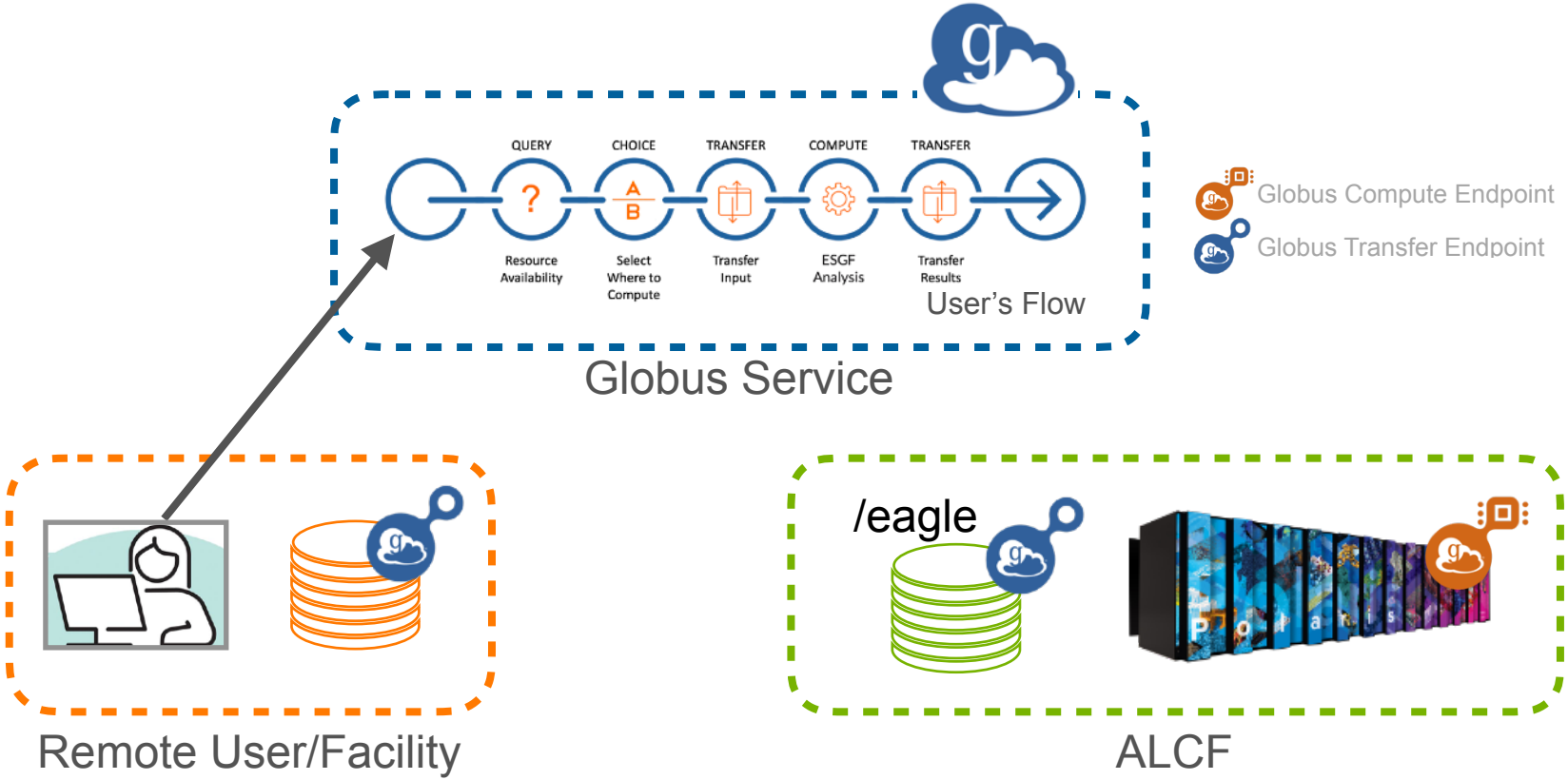
Globus Flows

Creating Workflows with Globus Compute Tasks

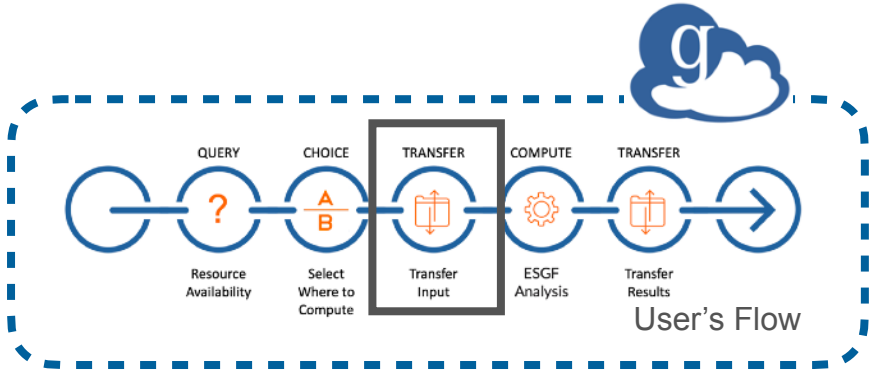
- Globus Flows is an automated and managed workflow service hosted by Globus
- Built on top of AWS step functions
- Hosted in the cloud in the Globus Service
- A ‘flow’ can couple actions of different types, e.g. a file transfer and a compute task



The Globus Flows Service

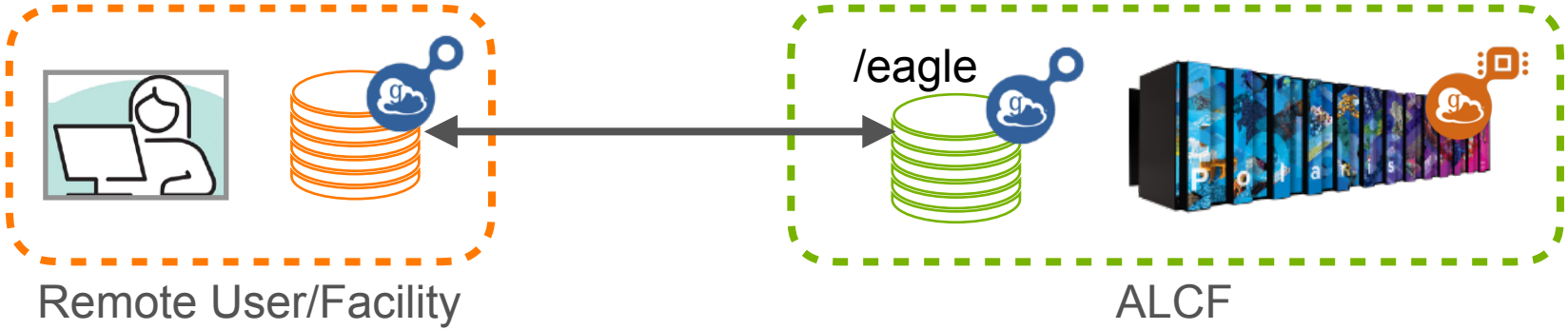


The Globus Flows Service

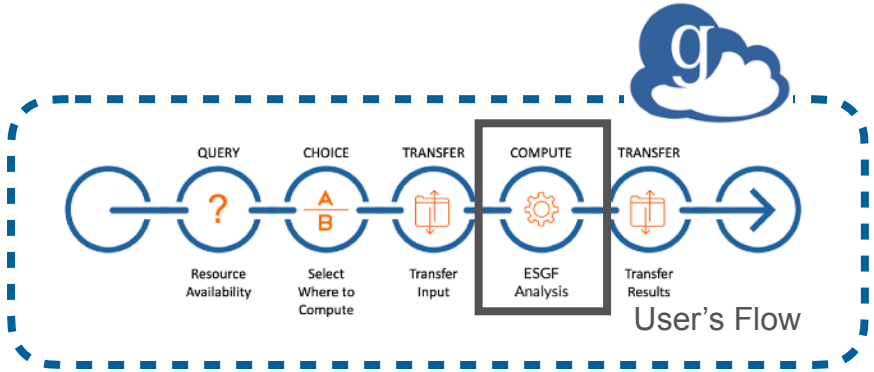


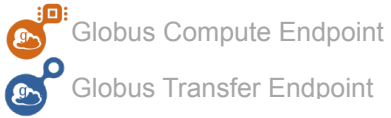
Globus Compute Endpoint

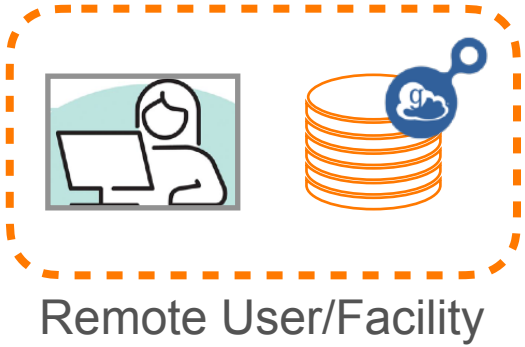
Globus Transfer Endpoint



The Globus Flows Service







Example Flow Actions

- **Transfer** - transfer files between two Globus collections
- **Compute** - execution of a function by a Globus compute endpoint
- **Choice** - an if/else decision that will move the flow to different actions depending on outcome
- **Delete** - deletes files from a Globus collection
- **Set Permissions** - changes permissions of files on Globus collection
- … and more

Example flows:

https://github.com/globus/globus-flows-trigger-examples/tree/main

Flow Definition

Example: 2 step transfer-compute flow

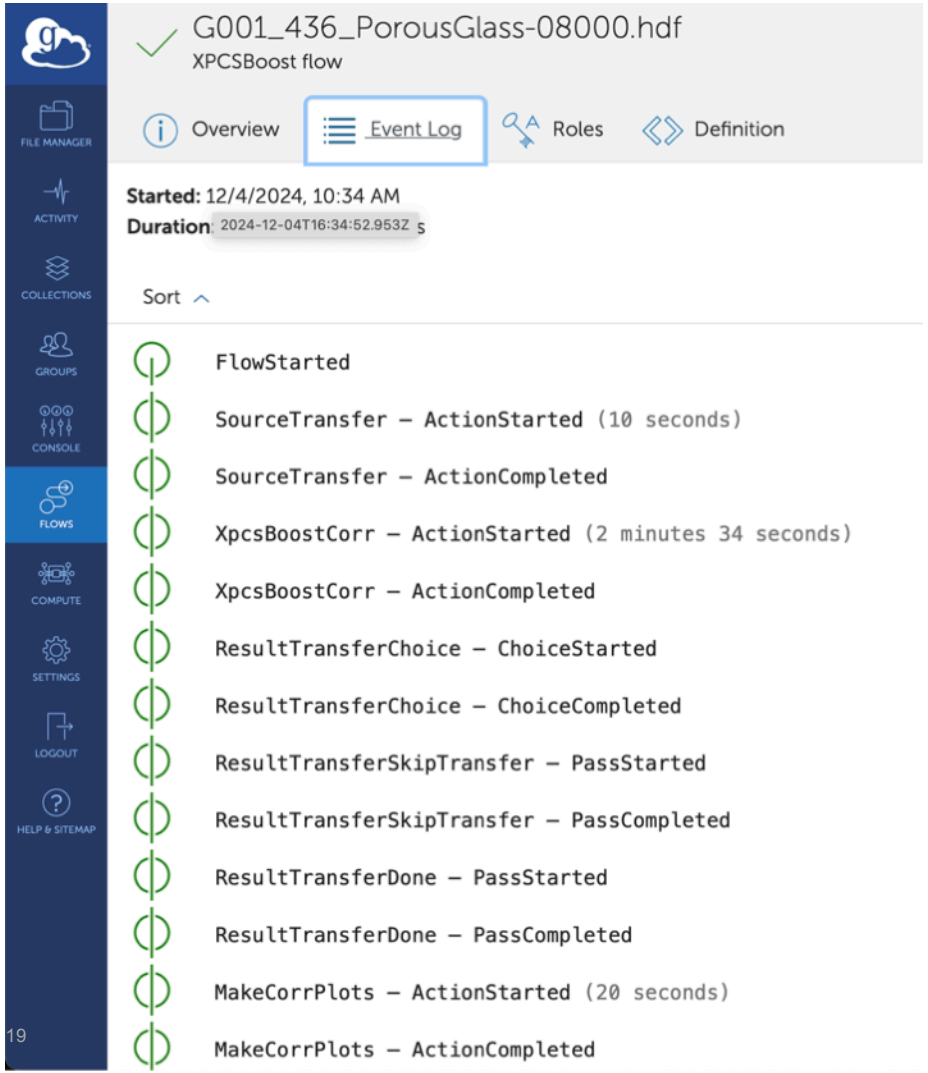
- Flow expressed as JSON object
- Two actions in this case
  - Transfer Action `TransferFiles`
  - Compute Action `ProcessFiles`
- Variables in the JSON denoted with $ syntax

```
{
    "Comment": "Transfer and process files by invoking a Globus Compute function",
    "StartAt": "TransferFiles",
    "States": {
        "TransferFiles": {
            "Comment": "Transfer files",
            "Type": "Action",
            "ActionUrl": "https://actions.automate.globus.org/transfer/transfer",
            "Parameters": {
                "source_endpoint_id.$": "$.input.source.id",
                "destination_endpoint_id.$": "$.input.destination.id",
                "transfer_items": [
                    {
                        "source_path.$": "$.input.source.path",
                        "destination_path.$": "$.input.destination.path",
                        "recursive.$": "$.input.recursive_tx"
                    }
                ]
            },
            "ResultPath": "$.TransferFiles",
            "WaitTime": 60,
            "Next": "ProcessFiles"
        },
        "ProcessFiles": {
            "Comment": "Process files - generate thumbnails",
            "Type": "Action",
            "ActionUrl": "https://compute.actions.globus.org",
            "Parameters": {
                "endpoint.$": "$.input.compute_endpoint_id",
                "function.$": "$.input.compute_function_id",
                "kwargs.$": "$.input.compute_function_kwargs"
            },
            "ResultPath": "$.ProcessFiles",
            "WaitTime": 180,
            "End": true
        }
    }
}
```

Monitoring Flow

Globus Web UI

Example link —>



Submitting Flow

A Brief Overview

- Steps to Submitting a Flow to the Globus Service
  - Register functions
  - Start Globus Compute endpoints
  - Register the Flow
  - Create a Flows Client
  - Submit Flows inputs

```
flow_input = {
    "input": {
        "recursive_tx": True,
        "source": {
            "id": source_transfer_endpoint_id,
            "path": "/src_collection/path/to/data"
        },
        "destination": {
            "id": dest_transfer_endpoint_id,
            "path": "/dest_collection/path/to/data"
        },
        "compute_function_id": process_files_function_id,
        "compute_endpoint_id": compute_endpoint_id,
        "compute_function_kwargs": {}
    }
}
run = specific_flow_client.run_flow(body=flow_input)
```

- See Documentation for details: https://globus-sdk-python.readthedocs.io/en/stable/services/flows.html
- Example Notebook with a transfer-compute flow: https://github.com/globus-labs/tomography_flow/blob/main/Tomography-flow.ipynb

Automation

Service accounts, clients, and secrets

- Users of Globus Flows are sometimes launching flows from remote services
- In these cases, ALCF offers service accounts that allow for easy maintenance of automated remote workflows
- Globus also offers automated authentication with client secrets (https://globus-sdk-python.readthedocs.io/en/stable/examples/client_credentials.html)

Conclusions

Remote Workflows at ALCF

- Execution of remote workflows are increasingly common on ALCF machines
- Users have had success using Globus Compute and Globus Flows to execute remote workflows at ALCF
- Globus compute is a “fire-and-forget” function execution service that will send work to Globus Compute Endpoints
- There are many ways of configuring Globus Compute Endpoints on Polaris depending on the resource needs of the application (similar to Parsl)
- Globus compute functions can be incorporated in Globus Flows. This allows for the coordination of compute tasks with data management tasks from the Globus service in the cloud
- Tools like ALCF service accounts and Globus confidential clients and secrets are available to facilitate automation for use cases that require it


Argonne
NATIONAL LABORATORY
ALCF